DETECTION AND EVALUATION OF MAJOR ERROR TRENDS OF SOFTWARE PROJECTS USING PARETO PRINCIPLE AND FUZZY MODEL

[1]Chandrakanth G Pujari, [2]Seetharam. K

[1]Associate Professor, Dept. of MCA
AIT, Bangalore, Research Scholar Sathyabama University Channai, India
[2]Professor and HOD, Dept. of ISE/CSE, SVCE, Bangalore
Email: [1]varuncp@rediffmail.com, [2]amoghks@rediffmail.com

Abstract

Pareto Principle is a statistical method to identify the minority of agents that extent the greatest effect. This principle promotes a win-win situation for the software projects, several organizations want to predict the number of errors in software systems, before they are deployed, to gauge the likely delivered quality and maintenance software. Finding errors in software is a challenging and time and budget consuming task. Minimizing these adverse effects using software error prediction models via guiding testers with defective parts of software system is an attractive research area. In this paper explores many aspects of the Pareto Principle and each aspect is related to detecting major error trends in projects, and examines software error prediction and improve prediction results. And also fuzzy model offers an easy-to-use tool for error evaluation in software projects. The model lies on fuzzy inference. The fuzzy model for error evaluation in software projects is an innovative instrument which can be used to forecast project failure. The model used to develop a software system for evaluating error in an e-testing project, so its applicability was validated.

Keywords: Software projects, Pareto Principle, fuzzy logic, Phase index, Error Index, frequency of errors,.

I. INTRODUCTION

J.M.Juran is the father of the Pareto Principle in quality management. He coined the terms vital few and trivial many as applied to the Pareto Principle. Fundamentally, the Pareto Principle stresses concentration on the vital few, not the trivial many. The historical development of the Pareto Principle illuminates its application to software quality.

Organizations are still asking how they can predict the quality of their software before it is used despite the substantial research effort spent attempting to find an answer to this question over the last 30 years[16][17][18]. One of the aims of software engineering activities is, cost effective development of high quality software systems, that is in the narrowest sense can be expressed as defect rate of final product[1][5]. Finding these defects before software have released is important. A defect found after delivery is usually more expensive than a defect found in development phase. Testing is the main activity of finding defects before software have released and it is the most challenging and time and budget consuming task of software life cycle[2][3][4].

We gather and analyze software data to help us to improve the software engineering process. As we face increasingly demanding software projects, we need to understand more precisely what we are doing and how to improve our effectiveness. This paper first outlines the principales of data gathering and then discusses the data gathering process. Data gathering is expensive and time-consuming. It affects the busiest people and may even be viewed as personally threatening. There is also considerable confusion an what data to gather and how to use it. While all these factors must be considered, there is no way to learn how to gather and analyze data without gathering and analyzing data[8][12].

1.1 The Process of Problem Analysis using Pareto Principle

In this analysis discusses the need for quantitative descriptions of software errors and methods for gathering such data. The software development cycle is reviewed and the frequency of the errors that are detected during software development and independent validation are compared. Data obtained from validation effort are presented[6][7], indicating the number of errors in 17 categories and three severity levels; the inferences that can be drawn from this data are discussed. Software development organization collects information on errors and defects for a period of one year. Some quality problems are uncovered as software is being developed[13][14][15]. Others are

encountered after the software has been released to its end user. Although hundreds of different errors are uncovered, all can be traced to the following 17 causes:

1. Erroneous data accessing (EDA)
2. Erroneous Arithmatic computations (EAC)
3. Invalid Timing (IT)
4. Improper Handling of Intrupts (IHI)
5. Wrong Constants and Data Value (WCDV)
6. Incomplete or Erroneous Specification (IES)
7. Misinterpretation of Customer Communication (MCC)
8. International Deviation from Specification (IDS)
9. Violation of Programming Standards (VPS)
10. Error in Data Representation (EDR)
11. Inconsistant Module Interface (IMI)
12. Error in Design Logic (EDL)
13. Incomplete or Erroneous testing (IET)
14. Inaccurate or Incomplete Documentation (IID)
15. Error in Programming Language Translation of Design ((PLT)
16. Ambiguous or Inconsistant Human-Computer Interface (HCI)
17. Miscellaneous (MIS).

For computer software, some data of frequency of occurrence or errors is available. First, Rubey's "Quantitative Aspects of Software Validation" data is presented. Table1 shows the basic Causes Error Categories for Software. Then for the major causes the common symptoms are shown in the Tables 2,3,4,5.

Software reliability personnel can draw several inferences from the data in Table3. First, there is no single reason for unreliable software, and no single validation tool or technique is likely to detect all types of errors. Second, the ability to demonstrate a program's correspondence to its specification does not justify complete confidence in the program's correctness, since a significant number of errors could be due to an incomplete or erroneous specification, and the documentation of the program cannot always be trusted. Third, intentional deviation from specification and the violation of established programming standards more often leads to minor errors than to serious errors. On the other hand, invalid timing or improper handling of interrupts almost always results in a significant error[19][20][21].

The data presented in Table 1 summarize the errors found in independent validations. In practice, however, the organization responsible for independent validation does not wait until the developer has completed program debugging. Instead, the independent validation organization often becomes involved at each program development phase to check that intermediate products(such as the program specification and program design) are correct. The errors occurring in the categorization of Table2, incomplete or erroneous specifications, indicate either deficiencies in, or the absence of, the verification of the program specification or program design, since there should be no errors in the final programs attributable to program specification if the preceding verification efforts were perfect.

As shown in Table2, 19 seriousand 82 moderate errors have escaped the verification efforts and have been found only during the checking of the actual coding. In 239 additional cases, errors due to incomplete or erroneous specification are considered of minor consequence; this is largely because the coding had been implemented correctly even though the program specification is itself in error.

If all of the 239 minor erroneous or incomplete specification errors were faithfully translated into coding, the total number of serious errors in the resultant coding would be 84 and the total number of moderate errors would be 162. Only 94 of the 239 minor errors would remain minor errors, even if the coding implemented the erroneous specification.This would make the incomplete or erroneous specification error category in Table 2 the largest error source by a factor of two, and would increase the total number of serious errors by 38 percent and the total number of moderate errors by 12 percent. Verification of the program specification and design in Advance of coding and debugging is a very beneficial activity, and indeed is probably essential if reliable software is desired.

To apply statistical SQA, Table 1 is built. In this table IES, MCC, EDA, EAC, and EDR are the vital few causes that accounted or 53 percent of all errors. It should he noted, however, that IES,EDR,PLT, and EDL would be selected as the vital few causes if only serious errors are considered. Once the vital few causes are determined, the software development organization can begin corrective action. For example, to correct MCC, the software developer might implement facilitated application To improve the quality

of customer communication and specification. To improve EDR, the developer might acquire specification techniques CASE tools for data modeling and perform more stringent data design reviews. It is important to note that corrective action focuses primarily on the vital few. As the vital few causes are corrected, new candidates pop to the top of the stack.

Table 1. Basic Causes Error Categories for Software

Error	Total		Serious		Moderate		Minor	
	NO.	%	NO.	%	NO.	%	NO.	%
EDA	120	8	36	16	72	12	12	2
EAC	113	7	22	10	73	12	18	3
IT	44	3	14	6	25	4	5	1
IHI	46	3	14	6	31	5	1	0
WCDV	41	3	14	6	19	3	8	1
IES	340	22	34	14	68	11	100	21
MCC	156	10	12	5	68	11	76	17
IDS	48	3	1	1	24	4	23	5
VPS	25	2	0	0	15	3	10	2
EDR	130	8	26	11	68	11	36	7
IMI	58	4	9	4	18	3	31	7
EDL	139	9	14	6	12	2	19	4
IET	95	6	12	5	35	6	48	10
IID	36	2	2	2	20	4	14	3
PLT	60	4	15	6	19	3	26	6
HCI	28	2	3	2	17	3	8	2
MIS	56	4	0	0	15	3	41	9
Totals	1535	100	228	100	599	100	476	100

Table 2. Common Symptoms for Software Errors: Incomplete or Erroneous Specifications

Error Category	Minor Error	Moderate Error	Serious Error	Total Error
Dimensional Error (DE)	17	17	7	41
Insufficient Precision Specified (IPS)	4	11	0	15
Missing Symbols or Labels (MS)	4	0	0	4
Typographical Error (TE)	51	0	0	51
Incorrect Hardware Description(IHD)	1	3	3	7
Design Consideration Incomplete or Incorrect(DCI)	122	47	8	177
Ambiguity in Specification or Design(ASD)	40	4	1	45
Total	239	82	19	340

Table 3. Common Symptoms for Software Errors: Erroneous Data Accessing

Error Category	Minor Error	Moderate Error	Serious Error	Total Error
Fetch or Store Wrong Data Word (FWDW)	10	52	17	79
Fetch or Store Wrong Portion of Data Word (FWPDW)	0	0	10	10
Variable Equated to Wrong Location (VEWL)	0	6	4	10
Overwrite of Data Word (ODW)	2	4	4	10
Register Loaded with Wrong Data (RLWD)	0	10	1	11
Total	12	72	36	120

Table 4. Common Symptoms for Software Errors: Erroneous Decision Logic or Sequencing

Error Category	Minor Error	Moderate Error	Serious Error	Total Error
Label Placed on Wrong Instruction/Statement (LPWI)	0	0	2	2
Branch Test Incorrect (BTI)	3	15	10	28
Branch Test Set Up Incorrect (BTSI)	0	1	1	2
Computations Performed in Wrong Sequence (CPWS)	6	2	1	9
Logic Sequence Incorrect (LSI)	6	65	27	98
Total	15	83	41	139

Table 5. Common Symptoms for Software Error :Erroneous Arithmetic Computation

Error Category	Minor	Moderate	Serious	Total
Wrong Arithmetic Operations Performed (WAOP)	10	47	12	69
Loss of Precision (LP)	2	6	1	9
Overflow (OF)	2	3	3	8
Poor Scaling of Intermediate Results (PSIR)	3	16	4	22
Incompatible Scaling (IS)	1	2	2	5
Total	18	73	22	113

III. ESTIMATION OF PHASE AND ERROR INDEX

In conjunction with the collection of defect information, we can calculate an Error index (EI) for each major step in the software engineering process. After analysis, design, coding, testing, and release, the following data are gathered: PI = w1*Si/Ei + w2*Mi/Ei + w3*Ti/Ei where Ei = the total number of errors uncovered during the ith step in the software engineering process, *Si* = the number of serious error, *Mi* = the number of moderate errors, *Ti* = the number of minor errors, *PS* = size of the product, *wj* = weighting factor, *j* = 1 to 3. The defective index (EI) is computed by calculating the cumulative effect or each PIi weighting errors encountered later in the software engineering process more heavily than those encountered earlier.

Table 6. Phase Index for Basic Error Categories for Software

PI(EDA)	5
PI(EAC)	4
PI(IT)	5
PI(IHI)	5
PI(WCDV)	5
PI(IES)	2
PI(MCC)	3
PI(IDS)	2
PI(VPS)	2
PI(EDR)	4
PI(IMI)	3
PI(EDL)	1
PI(IET)	3
PI(IID)	3
PI(PLT)	4
PI(HCI)	3
PI(MIS)	2

Error Index for Basic Error Categories for Software

$$EI = (\sum_{i=1}^{17} iP1i)/PS = 0.17 = 17\%$$

Table 7. Phase Index for Common Symptoms for Software Error :Incomplete or Erroneous

PI(DE)	3.36
PI(IPS)	2.47
PI(MS)	1
PI(TE)	1
PI(IHD)	5.72
PI(DC)	1.94
PI(ASD)	1.38

$$EI = (\sum_{i=1}^{7} iPIi)/PS = 0.19 = 19\%$$

Table 8. Phase Index for Common Symptoms for Software Error: Erroneous Data Accessing

PI(FWDW)	4.25
PI(FWPDW)	10
PI(VEWL)	5.8
PI(ODW)	5.4
PI(RLWD)	3.64

$$EI = (\sum_{i=1}^{5} iPIi)/PS = 0.68 = 68\%$$

Table 9. Phase Index for Common Symptoms for Software Error: Erroneous Decision Logic or Sequencing

PI(LPWI)	10
PI(BTI)	5.29
PI(BTSI)	8.5
PI(CPWS)	2.45
PI(LSI)	6.03

$$EI = (\sum_{i=1}^{5} iPIi)/ = 0.62 = 62\%$$

Table 10. Phase Index for Common Symptoms for Software Error: Erroneous Arithmetic Computation

PI(WAOP)	3.92
PI(LP)	3.33
PI(OF)	5.13
PI(PSIR)	4.01
PI(IS)	5.4

$$EI = (\sum_{i=1}^{5} iPIi)/PS = 0.61 = 61\%$$

IV. FUZZY MODEL FOR ERROR EVALUATION IN SOFTWARE PROJECTS

The proposed fuzzy model brings a solid contribution to error management by adapting existent techniques in errors evaluation to research projects. The model has important stages: error identification from an expert database, using building model components for fuzzy inference. The model allows error quantification by knowing the crisp values of error sources. Thanks to fuzzy logics mechanisms, the result has a higher estimation value[22][23].

4.1. Model Components for Fuzzy Inference

The proposed model for error evaluation in research projects has typical components of a fuzzy model: input variable, fuzzy rules[23]. The rules used in fuzzy risks modelling are built on the two well-known concepts from errors management (probability of error occurance and impact of it on project development) and an primary causes of errors. The model can be generally stated as: "The more over-budget is and the more embedded quality of the software development project idea, the lower the degree of error encountered in the software project."

Usually fuzzy models are used in decision making and they offer two types of answers: the error can be either accepted or rejected. The proposed model offers only a quantitative value of error, because the decision of accepting the error is taken by the human agent: project manager, errors manager or any other stakeholder. In conclusion, the output of the developed model isn't a form of decision, but an important parameter to make a proper decision[22]. The model components are further described, using fuzzy formalization.

4.2 Input Variables

The model has two forms of input variables: input functions and input constants. Input functions have the form of:

P(err) = probability of phase index error occurance

I(err) = impact of error index on software project

Where err = considered error code

They are described in Table 11, according to fuzzy logics concepts.

Table 11. Input Variables Description in Error Analysis Model

Fuzzy Variable Name	Universe of Discourse	Linguistic Grades
P(err)	[0,100]%	VL(very low), L(low), M(medium), H(high), VH(very high)
I(err)	[0,70]	VL(very low), L(low), M(medium), H(high), VH(very high)

Input constants have the form of:

ErrCause 1 = cause 1 of Err occurance

ErrCause 2 = cause 2 of Err occurrence

Where Err = considered error code

They are described in the same manner as input functions, the only difference being the defined universe of discource: it is specific to each identified cause.

4.3 Output Variables

The output variable is the value for an identified error and is denoted as: It is described in table 12

Table 12. Output variables description in error analysis model

Fuzzy variable name	Universe of Discourse	Linguistic Grades
V(Err)	[0,10]	VL(verylow), L(low), M(medium), H(high), VH(very high)

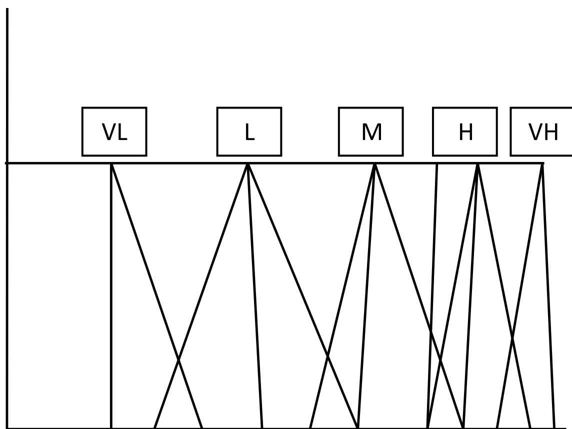

Fig. 1. Fuzzy Sets Representationm for Error Value in Software Projects

4.4 Model Rules

The error evaluation model consists from a set of predefined rules for establishing error value in software projects. These inference rules are mentioned in Table 13. The connective used to bind conditions in rules is "and". Besides the linguistic values of model variables (VL, L, M, H and VH), some restrictors are used:

- "somewhat" $= \sqrt[3]{\mu}$
- "very" $= \mu\, 2$

Where μ is the function showing if a numeric value belongs to a fuzzy set and it has values between 0 and 1 (a greater value shows a stronger membership).

Table 13. Inference Rules for Analysis Error "Err" in software projects

P(Err) /I(Err)	VL	L	M	H	VH
VL	VL	Somewhat VL	L	Somewhat M	Very H
L	L	L	L	H	VH
M	M	M	M	H	VH
H	H	H	H	Very H	VH
VH	VH	VH	VH	H	Very VH

If error Err has a high probability of occurance and the impact of this error is very high, then its value is also very high. (see the underlined values from table 13)

4.5 Formalization of Error Evaluation Model for E-testing Software Project

In order to compute the probability of "Lack of qualityresults" error (notated "P(Err)"), low embedded quality of the idea is reflected in "PhaseIndex" variable and over-budget sum in "ErrorIndex" variable. Both variables are defined in **table 14.** Inference rules for showing the effect of "LowQAIdea" and "OverBudget" changes on "Err" value are presented in **table 15**.

Table 14. Fuzzy variables for computing error in e-testing

Fuzzy Variable Name	Variable Type	Universe of Discourse	Linguistic Grades
PhaseIndex	Input	[0,10]	VL,L,M,H,VH
ErrorIndex	Input	[0,70]	VL,L,M,H,VH
P(Err)	input	[0,100]	VL,L,M,H,VH

Table 15. Inference rules for computing "P(Err)" in e-testing software project

Phase Ind / ErrIndex	VL	L	M	H	VH
VL	Somewhat M	Somewhat M	L	VL	Very VL
L	H	H	L	VL	VL
M	VH	H	M	L	L
H	VH	VH	H	M	M
VH	VH	VH	V H	M	M

V. SUMMARY AND CONCLUSION

Software process data is gathered to learn how to make process improvements. The principles of successful data gathering are: The data is gathered with a specific objective; the choice of data is based on a model of the process being examined; the data gathering process itself is defined and managed. It is tailored to the needs of the organization, and it must have management support. The data gathering plan specifies who will use the data and how it will be used. It covers why the data is needed, the data specifications, who will gather it and how, and how it will be validated and managed. A broad survey of the Pareto Analysis was presented. This included some historical perspective leading to the use of the Pareto Analysis as an effective quality tool for detecting major error trends during the development of software projects. Extensions of the Pareto Principle to software have been drawn from the areas of error identification, inspections, and statistical techniques.

The error index for Basic Error Categories for Software is less than the Common Symptoms for Software errors like Incomplete or Erroneous Specification, Erroneous Data Accessing, Erroneous Decision Logic or Sequencing and Erroneous Arithmetic Computation as estimated. The phase index and error index can be used in conjunction with information collected in Table 1,2,3,4,5 to develop an overall indication of improvement in software quality. We calculated phase index as shown in Table 6,7,8,9,and 10. The Pareto analysis can be summarized as experienced industry practitioners agree that most really difficult defects can be traced to a relatively limited number of root causes. In fact, most practitioners have an intuitive feeling for the "real" causes of software quality problems, but few have spent time collecting data to support their feelings. The vital few causes for errors can be isolated and appropriate corrections can be made. The proposed model offers an easy-to-use tool for error evaluation in software projects. Software projects are known for their high level of error, very few dedicated error systems were developed especially for them. Therefore, the fuzzy model for error evaluation in software projects is an innovative instrument which can be used to forecast project failure. The model was used to develop a software system for evaluating error in an e-testing software project, so its applicability was validated. The system can be further developed to evaluate all errors, not only the one from the highest level, as it does now.

REFERENCES

[1] Sommerville; Software Engineering, Pearson Education, 2007.

[2] K.O.Elish and M.O.Elish; Predicting defect-prone software modules using support vector machines, The Journal of Systems and Software, Vol. 81, 2008, pp. 649-660.

[3] C. Catal, and B. Diri,; A Systematic review of software fault prediction studies, Expert Systems with Applications, Vol. 36, 2009, pp. 7364-7354.

[4] A .D. Oral and A.B. Bener; Defect Prediction for Embedded Software, in 22nd International Symposium on Computer and Information Science, 2007, pp. 1-6.

[5] B.Turhan and A.Bener; Analysis of Naïve Bayes assumptions on software fault data; An empirical study Data & Knowledge Engineering, Vol. 68, 2009, pp. 278-290.

[6] T. Menzies and J. Greenwald; Data Mining Static Code Attributes to learn Defect Predictors, IEEE Transactions on Software Engineering, Vol. 33, 2007, pp. 2-13.

[7] N.E.Fenton and M.Neil; A Critique of Software Defect Prediction Models, IEEE Transactions on Software Engineering, Vol. 25, issue 5, 1995, pp. 675-689.

[8] Garmus. David & Herron. David; Measuring the Software Process: APractical Guide to Functional Measurements: Prentice Hall, Englewood Cliffs, NJ; 1995.

[9] Capers. Jones; Applied Software Measurement: 3rd edition; McGraw-Hill, New York, NY, 2008.

[10] Capers. Jones; Estimating Software Costs; 2nd edition; McGraw-Hill, New York, NY, 2007.

[11] Kan. Stephen H.; Metrics and Models in Software Quality Engineering, 2nd edition: Addision Wesley Longman, Boston, MA; ISBN 0-201-72915-6; 2003; 528 pages.

[12] Basili,V.R., and D.M.Weiss. "A Methodology for collecting valid software engineering data, "IEEE Transactions on Software Engineering, vol. SE-10, no. 6, nov 1984.

[13] Card, and Clerk, R.A.Berg. "Improving software quality and productivity", Information and Software Technology, vol.29, no. 5,June 1987.

[14] R.J.Rubey, J.A.Dana, and P.W.Biche, "Quantitative aspects of software validation", IEEE Transactions on Software Engineering, vol. SE-1, no. 2, June 1975.

[15] Endres.A, Rombach.D, A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories, Pearson Addition Wesley, 2004.

[16] Basili.V.R, and Green.S, Software Process Evolution at the SEL, "IEEE Software, July 1994, pp. 58-66.

[17] Jones.C Applied Software Measurement 1996.

[18] Kearney,J.K., R.L.Sedlmeyer,W.B.Thompson, M.A.Gray, and M.A.Adler, "Software complexity measurement," Communications of the ACM, vol.29, no. 11, Nov 1986.

[19] The Wall Street Journal, November 13, 1987.

[20] carson, "Scrap, Defect Rates Cut by 50%," In Quality, vol. 30, no. 4, April 1996.

[21] Humphrey, Watts, "Making Software Manageable." In CrossTalk, 1996, pp. 3-6.

[22] Hussain, O.K.Chang. etal. "A Fuzzy Approach to Risk Based Decision Making", Lecture Notes in Computer Science, Springer Berlin/Heidelberg, vol. 4278, 2006, pp. 1765-1775

[23] Teodorescu, H,N,Zbancioc.Metal. "knowledge Based Systems Applications, performantica, lassy, 2004